การปรับค่าน้ำหนักสำหรับโครงข่ายประสาทเทียมชนิดจัดกลุ่มเอง ในปริภูมินอกแบบยุคลิด

นายสายชล ใจเย็น

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาการคณนา ภาควิชาคณิตศาสตร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2546
ISBN 974-17-4281-9

WEIGHT ADJUSTING FOR A SELF-ORGANIZING ARTIFICIAL NEURAL NETWORK IN NON-EUCLIDEAN SPACE

Mr. Saichon Jaiyen

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computational Science

Department of Mathematics
Faculty of Science
Chulalongkorn University

Academic Year 2003

ISBN 974-17-4281-9

Thesis Title	WEIGHT ADJUSTING FOR A SELF-ORGANIZING ARTIFICIAL NEURAL NETWORK IN NON-EUCLIDEAN SPACE
By	Mr. Saichon Jaiyen
Field of Study	Computational Science
Thesis Advisor	Professor Chidchanok Lursinsap, Ph.D.

Accepted by the Faculty of Science, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

.....Dean of the Faculty of Science

(Professor Piamsak Menasveta, Ph.D.)

THESIS COMMITTEE

.....Chairman

(Assistant Professor Wicharn Lewkeeratiyutkul, Ph.D.)

.....Thesis Advisor

(Professor Chidchanok Lursinsap, Ph.D.)

.....Member

(Associate Professor Suchada Siripant)

.....Member

(Paisan Nakmahachalasint, Ph.D.)

.....Member

(Nataphan Kitisin, Ph.D.)

สายชล ใจเย็น : การปรับค่าน้ำหนักสำหรับโครงข่ายประสาทเทียมชนิดจัดกลุ่มเองในปริภูมินอกแบบยุคลิด (WEIGHT ADJUSTING FOR A SELF-ORGANIZING ARTIFICIAL NEURAL NETWORK IN NON-EUCLIDEAN SPACE) อาจารย์ที่ปรึกษา : ศาสตราจารย์ ดร. ชิดชนก เหลือสินทรัพย์, 57 หน้า. ISBN 974-17-4281-9.

การแบ่งกลุ่มข้อมูลโดยใช้วิธีการเรียนรู้แบบไม่มีผู้สอนนั้น ในปัจจุบันจะใช้วิธีการเรียนรู้ด้วยตัวเองที่มีการแข่งขันกันระหว่างเซลล์ประสาทเทียมและข้อมูลที่ถูกเลือกโดยมีพื้นฐานบนการใช้ระยะทางแบบยุคลิด เนื่องจากการวัดระยะทางแบบยุคลิดมิได้พิจารณาลักษณะโครงสร้างของข้อมูลที่จะแบ่งกลุ่ม ดังนั้นวิธีการนี้จึงไม่เหมาะสมในการนำมาใช้ในปัญหาการแบ่งกลุ่มข้อมูลในกรณีที่มีโครงสร้างทางเรขาคณิตและพื้นผิวของปริภูมิของข้อมูลมาเกี่ยวข้องด้วย การเคลื่อนที่ของเวกเตอร์ตัวแทนของกลุ่มข้อมูลควรจะเคลื่อนที่ไปตามความโค้งของปริภูมิข้อมูลและแบ่งกลุ่มข้อมูลอย่างถูกต้อง เพราะฉะนั้นระยะทางที่ใช้วัดความคล้ายกันของข้อมูลควรจะเป็นระยะทางที่ไม่ใช่แบบยุคลิด แต่เป็นระยะทางที่วัดตามโครงสร้างจริงของปริภูมิข้อมูล ปัญหาที่จะศึกษาในงานวิจัยนี้จะเกี่ยวข้องกับ วิธีการวัดระยะทางที่ไม่ใช่ระยะทางแบบยุคลิดในปริภูมิข้อมูล ตัวอย่างเช่นบนพื้นผิวที่ไม่รู้ฟังก์ชันของพื้นผิว และการเคลื่อนที่ของเวกเตอร์ตัวแทนของกลุ่มข้อมูลไปตามโครงสร้างทางเรขาคณิตที่แท้จริงของปริภูมิข้อมูล วิธีการนี้ประสบความสำเร็จอย่างมากในการแบ่งกลุ่มข้อมูลทดลองในหลายรูปแบบ ในขณะที่การแบ่งกลุ่มข้อมูลที่ใช้ระยะทางแบบยุคลิดโดยใช้วิธีการเรียนรู้แบบไม่มีผู้สอนนั้นให้ผลที่ไม่ถูกต้อง

ภาควิชา	**คณิตศาสตร์**	ลายมือชื่อนิสิต.....
สาขาวิชา	**วิทยาการคณนา**	ลายมือชื่ออาจารย์ที่ปรึกษา.....
ปีการศึกษา	2546	

437 24420 23 : MAJOR COMPUTATIONAL SCIENCE
KEYWORD: SOM IN NON-EUCLIDEAN SPACE / CLUSTERING / FEATURE EXTRACTION / DATA COMPRESSION

SAICHON JAIYEN : WEIGHT ADJUSTING FOR A SELF-ORGANIZING ARTIFICIAL NEURAL NETWORK IN NON-EUCLIDEAN SPACE. THESIS ADVISOR : PROFESSOR CHIDCHANOK LURSINSAP, Ph.D., 57 pp. ISBN 974-17-4281-9.

Current unsupervised classification using self-organizing competitive learning is based on the minimum Euclidean distance between a prototype neuron and the selected data. Euclidean distance measure does not consider the clustering structure of the data. This approach is not suitable for several classification problems where the geometrical structure and surface of the data space are the main concern. The prototype neurons must flow along the natural curvature of the data space and correctly classify the space. This implies that the distances among the data should be non-Euclidean and measured along the natural structure of the space. The problem studied in this paper concerns the algorithm for measuring the non-Euclidean distance in a data point space, i.e. the surface function is unknown, and moving the prototype neurons along the actual geometrical structure of the data points. Our algorithm successfully classifies the experimental data spaces with various aspects while the unsupervised Euclidean distance classification gives incorrect results.

Department **Mathematics** Student's signature……...
Field of study **Computational Science** Advisor's signature……..
Academic year **2003**

Acknowledgements

I am greatly indebted to my thesis advisor, Prof. Dr. Chidchanok Lursinsap, for kindly providing guidance throughout the development of this thesis. His comments greatly helped me in all the time of research work. What I know today about the good process of research , I learned from him. I also extend my sincere gratitude and appreciation to Assoc. Prof. Suchada Siripant for her guidance during my education. Furthermore, I would like to special thanks to Assistant Professor Dr. Wicharn Lewkeeratiyutkul, Dr. Nataphan Kitisin, and Dr. Paisan Nakmahachalasint for their valuable suggestions.

I want to express my gratitude to King Mongkut's Institute of Technology Ladkrabang for their scholarship in supporting me during my education.

I am grateful to The Advanced Virtual and Intelligence Computing Research Center (AVIC) for their material support in enabling me to accomplish this research.

I would like to thank Anuchit, Ekavit, Denpong, Maytee, Tomeyot, San, Bancha, Apichat and others at AVIC for the valuable discussion. I would also like to thank my friends in the Department of Mathematics for providing me constant encouragement.

I feel a deep sense of gratitude for my parents who taught me the good things that really matter in life and give me the love and encouragement.

Finally, I would like to thank all whose direct and indirect support helped me completing my thesis.

Table of Contents

Table of Contents (continued)

List of Figures

List of Figures (continued)

CHAPTER 1

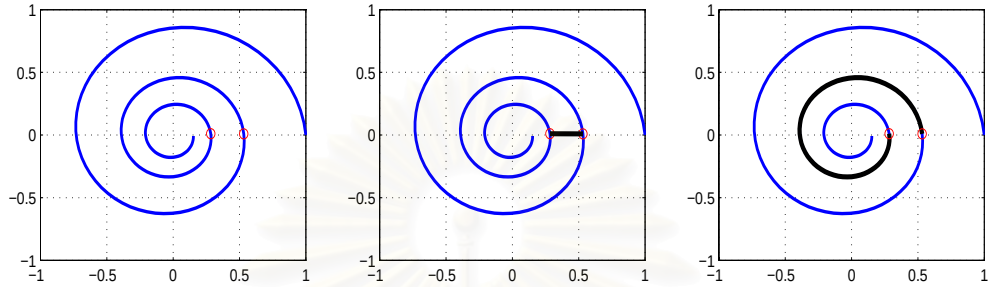
INTRODUCTION

1.1 Problem Identification

Kohonen's Self-Organizing Map (SOM) neural network [1] is a type of unsupervised neural model, which is not necessary to specify the targets in the training data. The model partitions the data into the clusters. The SOM neural network is broadly applied to several industrial problems such as pattern classification, feature extraction, data compression, data mining, and many more. In addition, it may be used to quantize large data by grouping the data into a number of clusters and representing each groups by a prototype neuron.

While the applications of the self-organizing map (SOM) neural network are extremely widespreaded, most proposed SOM learning models such as hierarchical variants [2], the ASSOM [3], the GTM [4], the ESOM [6] and the stochastic SSOM for mapping of proximity data [5] still employ Euclidean distance to measure the similarity among the data. Furthermore, the training algorithms must specify the number of clusters in advance. These constraints are not suitable for many classification problems where the classes are defined by the natural curvature of the data space. Consider the example shown in Figure 1.1. The data points are scattered and formed a spiral shape in a 2-dimensional space. Two different

measures are used to measure the distances between these two points. Euclidean distance is shown in Figure 1.1(b) but the natural manifold distance is used in Figure 1.1(c). It is obvious that Euclidean measure gives an incorrect classification but the natural manifold distance correctly classifies the data points.



(a) (b) (c)

Figure 1.1: An example of different distance measures. All data points are scattered and formed a spiral shape in a 2-dimensional space. (a) Two random points in a spiral. (b) Euclidean distance between the two points. (c) The natural manifold distance between the points.

In the classical SOM learning, the prototype neuron does not move on the natural surface or inside the *cloud* of data but it always moves toward the training data point in the direction defined in terms of Euclidean distance between the prototype location and the data point. Actually, the prototype neuron must be moved along the surface or the manifold of the data point space. If the surface or the manifold of the data space can be captured in forms of a function then the problem of moving all prototype neurons becomes simple. The concept of a Reimanian distance and tensor can be directly applied. But in the actual applications, it is impossible to formulate a surface function. Therefore, the problem

is how to estimate the natural manifold distance between two data points.

In this research, we concentrate on scattered data on a surface or a manifold due to the fact that many data are regarded as a discrete surface or a manifold. For example, the surface of a 3-D image is formed by a set of unorganized cloud points representing surface samples from laser range scanners. The goal of this research is to develop a new Self-Organizing Maps learning algorithm using a non-Euclidean distance on a surface to identify the winning neuron and to move the synaptic weight vectors of the neurons along the surface.

1.2 Objective and Contributions

The goal of this research is to develop a new Self-Organizing Maps learning algorithm using a non-Euclidean distance on a surface to identify the winning neuron and to move the synaptic weight vectors of the neurons along the surface. The algorithm developed can be applied to several industrial problems such as feature extraction, pattern classification, and data compression.

1.3 Scope of Work and Constraints

1. The data are scattered and formed a discrete surface or a manifold.
2. The surface function is not known and not defined in advance.

1.4 Outline of the Thesis

The remaining contents are organized into six chapters. Chapter II reviews the related literatures. Chapter III gives a review of Self-Organizing Map. Our new Self-Organizing Map on a non-Euclidean space model and the algorithm are proposed in Chapter IV. The experimental results and a comparison between this experimental results and the results in [1] are given in Chapter V. Chapter VI concludes the research.

CHAPTER 2

LITERATURE REVIEW

Ritter [10] proposed a new type of Self-Organizing Map (SOM) called *hyperbolic SOM* (HSOM) based on a regular tessellation of the hyperbolic plane. It is a non-Euclidean space characterized by constant negative Gaussian curvature. The HSOM uses the hyperbolic lattice topology for the arrangement of the HSOM nodes. The number of nodes in the lattice grows very rapidly with the chosen lattice radius r. Each lattice node carries a prototype neuron from a high dimensional feature space. Demartines and Herault [7] presented a new strategy called *curvilinear component analysis* (CCA) for dimensionality reduction to improve Kohonen's self-organizing map. The CCA algorithm, which borrows both the ideas of multivariate data analysis and the principle of SOM, proceeds in two steps: vector quantization (VQ) of the sub-manifold in the input space and nonlinear projection of these quantizing vectors toward an output space, providing a revealing unfolding of the sub-manifold. In addition, the projection of CCA is similar to other nonlinear mapping methods such as multidimensional scaling (MDS) and Sammon's nonlinear mapping (NLM). The CCA has been successfully applied to various nonlinear problems of data representation in the framework of process surveillance, sensor fusion, and generation of metric spaces from non-metric cost functions. Bishop and Svensen [4] introduced a form of non-linear

latent variable model called the Generative Topographic Mapping for which the parameters of the model can be determined using the EM algorithm. GTM provides a principle alternative to the widely used Self-Organizing Map (SOM) of Kohonen and overcomes the most significant limitations of the SOM such as the absence of a cost function, the lack of a theoretical basis for choosing learning rate parameter schedules, and neighborhood parameters to ensure topographic ordering. The GTM model is defined in terms of a mapping from the latent space into the data space. The GTM algorithm uses the batch version which all of the training data are used together to update the model parameters. Deng and Kasabov [6] proposed a dynamic version of the Kohonen's Self-Organizing Map (SOM) called "Evolving Self-Organizing Maps" (ESOM). The network structure of ESOM is different from the standard SOM. The network structure is not fixed and the prototype nodes are not organized onto one or two-dimensional lattices. The ESOM network starts without any node and some new nodes are created during learning. Each node carries a weight vector of the same dimensionality as the input data. The connections between the mapped nodes are used to maintain the neighborhood relationships between the close nodes. The strength of the neighborhood relation is determined by the distance between those connected nodes. If the distance is too large, the connection can be pruned.

CHAPTER 3

BACKGROUND KNOWLEDGE

3.1 An Overview of Artificial Neural Networks

The human brain contains many different specialized information centers scattered throughout its three-dimensional space. The most basic component of the human brain is a specific type of cells, which provides us with the abilities to remember, think, and apply previous experiences to our every action. These cells are known as neurons. Each of these neurons can connect with up to 200000 other neurons. All biological neurons have four basic components, which are dendrites, soma (or cell body), axon, and synapses [11] as shown in Figure 3.1.

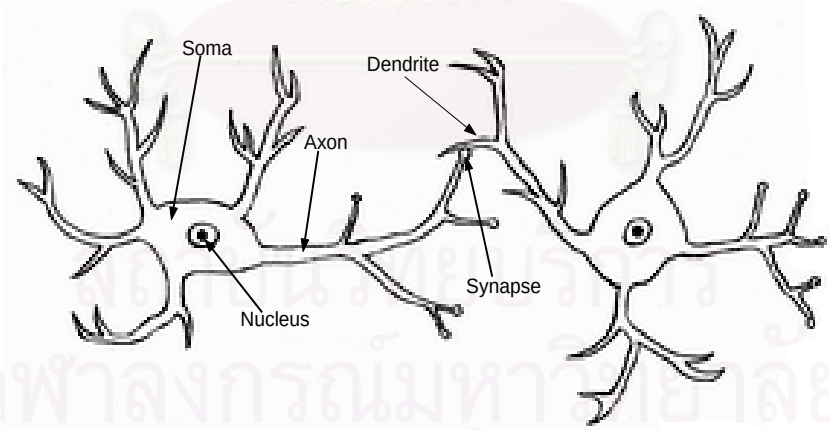


Figure 3.1: Two interconnected biological cells.

Learning in biological systems involves the adjustments of the synaptic connections that exist between the neurons. This is also true for artificial neural networks .

Artificial neural networks are collections of mathematical models that emulate some properties of the human brain. Each artificial neuron is designed to have the same components similar to a biological neuron in the human brain as follows:

Soma corresponds to a neuron node.

Axon corresponds to an output.

Dendrite corresponds to an input.

Synapse corresponds to a weight.

There are three categories of learning algorithms which are:

1. Supervised learning: A neuron is forced to generate an output signal equal to a specific target signal of an input pattern and to reproduce this target signal whenever the specific input pattern appears.

2. Unsupervised learning: It must not specify a target signal associated with a specific input pattern. A neural competitively adjusts its weight similar to the input patterns.

3. Reinforcement learning: It is a hybrid between supervised and unsupervised learnings under the environment which the target cannot be explicitly defined.

3.2 A Self-Organizing Map

The self-organizing map (SOM) is the type of unsupervised learning model. The principal goal of the self-organizing map is to transform an incoming signal pattern into a one- or two-dimensional discrete map, and to perform this transformation adaptively in a topologically ordered fashion. This network usually contains neurons arranged in rows and columns in a lattice and each neuron is completely connected to all the source nodes in the input layer. A one-dimensional lattice comprises only of a single column or row of neurons. Figure 3.2 shows a two-dimensional lattice of neurons commonly used as the discrete map.

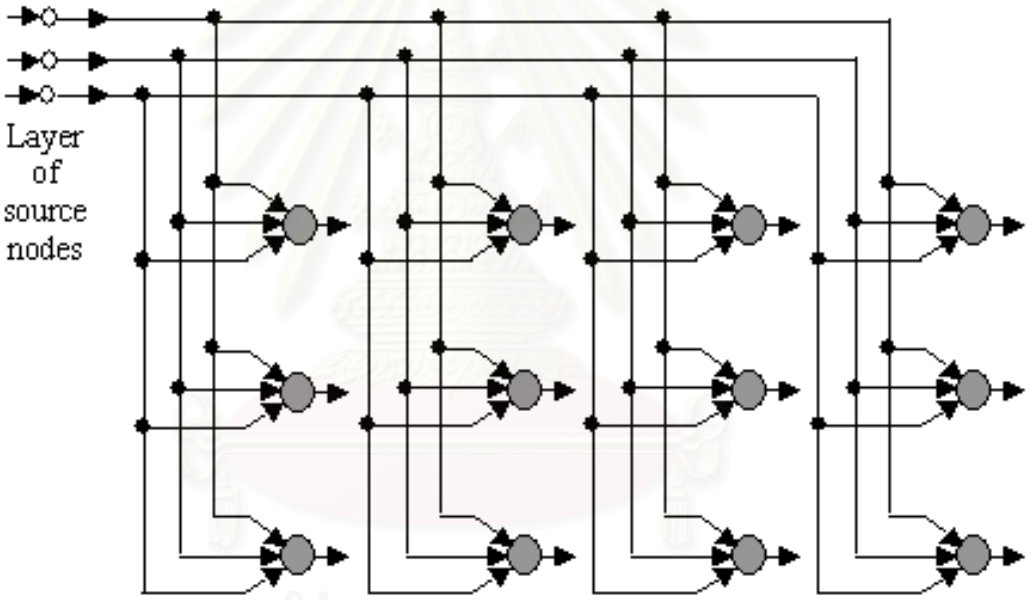


Figure 3.2: The network of the self-organizing map.

The learning algorithm for the self-organizing map commence with initializing the synaptic weights in the network. After the network has been appropriately initialized, there are three essential processes involved in the learning algorithm

of the self-organizing map:

1. Competition: Each neuron competes with each other to be a winner. A winner is a neuron whose weight vector and the currently learned input vector have a shortest Euclidean distance among the distances between the currently learned input vector and other weight vectors. Only the weight of the winner is updated.

2. Cooperation: Not only the winning neuron will have its weights updated, but its close neighbors will also have the chance to update their weights, although not as much as the winning neuron.

3. Synaptic Adaptation: The weights should be updated in such a way that the new weights will be closer to the input.

The details of the processes of competition, cooperation, and synaptic adaptation are now demonstrated.

3.2.1 Competitive Process

Let m be the dimension of the input space. Let an input vector selected at random from the input space be denoted by

$$\mathbf{x} = [x_1 \; x_2 \; \dots \; x_m]^{\mathrm{T}} \tag{3.1}$$

The synaptic weight vector of each neuron in the network has the same dimension as the input space. Let the synaptic weight vector of neuron j be denoted by

$$\mathbf{w}_j = [w_{j1} \; w_{j2} \; \dots \; w_{jm}]^{\mathrm{T}}, \qquad j = 1, \dots, k \tag{3.2}$$

where k is the total number of neurons in the network.

We use the Euclidean distance to find the best match of the input vector $\mathbf{x}$ with respect to the synaptic weight vector $\mathbf{w}_j$.

If we use the index i^* to identify the neuron that best matches the input vector $\mathbf{x}$, we may then determine i^* by applying the condition:

$$i^* = \arg\min_j \|\mathbf{x} - \mathbf{w}_j\|, \quad j = 1, \ldots, k. \tag{3.3}$$

The particular neuron i that satisfies this condition is called the best-matching or the winning neuron for the input vector $\mathbf{x}$.

Depending on the interested application, the response of the network could be either the index of the winning neuron (i.e., its position in the lattice), or the synaptic weight vector that is closest to the input vector in a Euclidean sense.

3.2.2 Cooperative Process

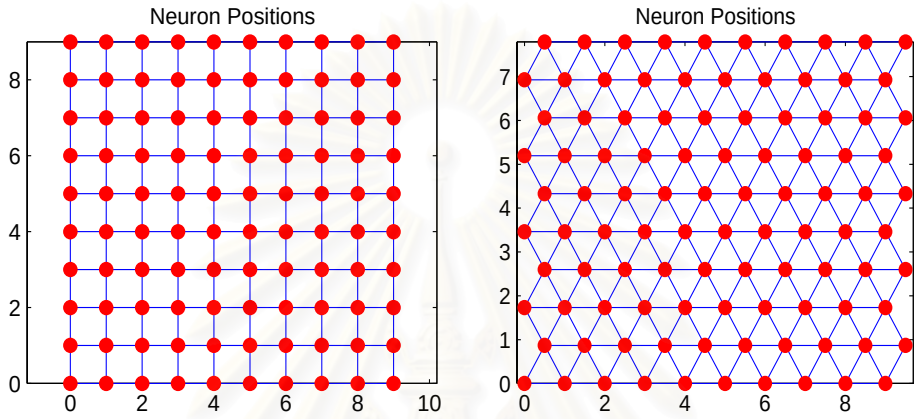
In neurobiological studies, there exists the lateral interaction among a set of exited neurons. In particular, a neuron that is firing tends to excite the neurons in its immediate neighborhood more than those farther away from it. There is a topological neighborhood that decays with distance. This neurobiological evidence defines a resembling topological neighborhood for the neurons in the SOM.

Let $d_{j,i}$ be the lateral distance between winning neuron i^* and excited neuron j, we define

$$h_{j,i^*} = \exp\left(-\frac{d_{j,i^*}^2}{2\sigma^2}\right) \tag{3.4}$$

as the topological neighborhood, where i^* is the index of the winning neuron and σ is the width of the topological neighborhood. The topological neighborhood

h_{j,i^*} depends on lateral distance d_{j,i^*} which is the Euclidean distance between winning neuron i^* and excited neuron j measured in the output space. The distance between neurons are usually calculated from their positions in a row-dimensional rectangular or hexagonal grid as demonstrated in Figure 3.3 (a) and (b), respectively.



(a) (b)

Figure 3.3: (a) The positions of 100 neurons arranged in rectangular grid. (b) The positions of 100 neurons arranged in hexagonal grid.

For the special feature of SOM, the size σ of the topological neighborhood needs to decrease with time. A popular time dependence is the exponential decay described by

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right), \quad n = 0, 1, \ldots, \tag{3.5}$$

where σ_0 is the value of σ at the initiation of the SOM algorithm, and τ_1 is a time constant. We substitute Eq.(3.5) in Eq.(3.4) and obtain

$$h_{j,i^*}(n) = \exp\left(-\frac{d_{j,i^*}^2}{2\sigma^2(n)}\right), \quad n = 0, 1, \ldots, \tag{3.6}$$

where $\sigma(n)$ is defined by Eq.(3.5). Henceforth we will refer to $h_{j,i^*}(n)$ as the **neighborhood function**. The goal of the neighborhood function is essentially to correlate the directions of the weight adjusting of a large number of excited neurons in the lattice.

3.2.3 Adaptive Process

For the network to be self-organizing, the synaptic weight vector $\mathbf{w}_j$ of neuron j in the network should be updated in such a way that the new weights will be closer to the input vector $\mathbf{x}$. Concerning weight adjusting, not only the winning neuron gets its weight updated, but its close neighbors will also have updated their weights, although not as much as the winner.

Given the synaptic weight vector $\mathbf{w}_j(n)$ of neuron j at time n, the updated weight vector $\mathbf{w}_j(n+1)$ at time $n+1$ is defined by

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n)h_{j,i^*}(n)(\mathbf{x} - \mathbf{w}_j(n)) \tag{3.7}$$

where $\eta(n)$ is a learning rate depending on time as indicated by

$$\eta(n) = \eta_0 \exp\left(-\frac{n}{\tau_2}\right), \quad n = 0, 1, \ldots, \tag{3.8}$$

where τ_2 is another time constant and η_0 is the value of η at the initiation of the SOM algorithm.

The effect of the weight adjusting is to move the synaptic weight vector $\mathbf{w}_j$ of winning neuron j and its neighbors toward the input vector $\mathbf{x}$.

3.3 Summary of SOM Algorithm

All neurons are organized in a two dimensional array and their physical locations in the input data space are represented by their weight vectors. There are four essential steps concerning the SOM learning process: initialization: sampling, similarity matching, and updating. These four steps are iterated until the formation of the feature map has completed. Let $\mathbf{w}_j(t)$ be the prototype neuron j at time t. The dimension of each prototype neuron is equal to the dimension of the data space. The stages of SOM algorithm can be summarized as follows:

1. **Initialization**. Randomly set the values of all prototype neurons $\mathbf{w}_j(0)$, $1 \leq j \leq k$, where k is the number of neurons in the lattice. Another way of initializing is to select the prototype neurons $\{\mathbf{w}_j(0)\}_{j=1}^{k}$ from the available set of data vectors $\{\mathbf{x}_i\}_{i=1}^{N}$ in a random manner.

2. **Sampling**. Draw a sample training data vector $\mathbf{x}_i$ from the data space.

3. **Similarity Matching**. Find the winning neuron $\mathbf{w}_{i*}(t)$ with respect to data vector $\mathbf{x}_a$ by using the minimum-distance Euclidean criterion:

$$||\mathbf{w}_{i*}(t) - \mathbf{x}_a|| = \min_j (||\mathbf{x}_a - \mathbf{w}_j(t)||), \quad 1 \leq j, i* \leq k. \tag{3.9}$$

4. **Updating**. Adjust the synaptic weight vectors of all neurons at time t by using the updating formula:

$$\mathbf{w}_j(t) = \mathbf{w}_j(t-1) + \eta(t-1)h_{j,i}(t-1)(\mathbf{x}_a - \mathbf{w}_j(t-1)). \tag{3.10}$$

5. **Continuation**. Return to step 2 until the feature map stops changing.

3.4 An Example of Self-Organizing Mapping Neural Network

This example shows how a two-dimensional self-organizing map can be trained using a network with 100 neurons arranged in 10 rows and 10 columns. First, 1000 random input data were created as demonstrated in Figure 3.4(a). Second, the initial values of the synaptic weights are chosen randomly as presented in Figure 3.4(b). Then, we set the initial values of the learning-rate parameter and set the initial values of the neighborhood size . After that we start training each of the input vector. During the ordering phase, the map has begun to organize itself to form a mesh, as shown in Figure 3.4(c). Finally, after 200 epochs, the map spreads out to fill the input space as given in Figure 3.4(d).

3.5 The batch version of the SOM

The batch computation version is significantly faster. If all samples $\{\mathbf{x}_i\}_{i=1}^{N}$ are the available set of data vectors, they can be trained by batch algorithm. The batch algorithm can be summarized as follows:

1. Initialize the values for the prototype neurons $\mathbf{w}_i$ in some proper way.
2. For each map unit j, collect a list of all data vectors $\mathbf{x}_i$, whose most similar prototype neuron $\mathbf{w}_j$ belongs to the neighborhood set N_j of node j.
3. Compute the mean of the vectors $\mathbf{x}_i$ in each N_j and replace the old values $\mathbf{w}_j$ by the corresponding mean.
4. Repeat from step 2 until the values of all prototype neurons are not changed.

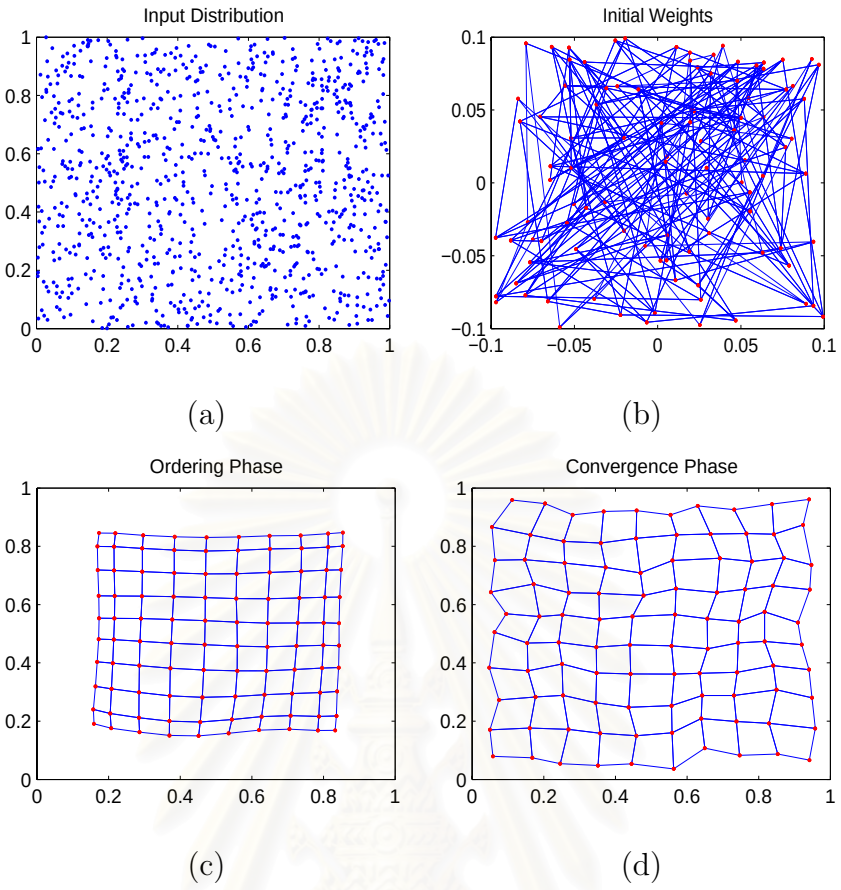


Figure 3.4: (a) A plot of 1000 input vectors. (b) 100 initial synaptic weight vectors. (c) The ordering phase. (d) The convergence phase.

Notice that step 3 is to calculate the mean over the union of the lists that belong to the neighborhood set N_j of unit j.

CHAPTER 4

Mathematical Model and Algorithm

4.1 Mathematical Model for Self-Organizing Artificial Neural Network in a Non-Euclidean Space

The principal of unsupervised learning is to measure the similarity among input data and to partition them into a set of clusters. The Euclidean distance is one of the most popular measures used in this unsupervised learning. In fact, the similarity between two input data measured by the Euclidean distance is based on the direct coordinate distance between these input data considered as two vectors in a high dimensional space. The actual geometrical structure of data space is not involved during this similarity measure. Generally, the input data are scattered and formed a discrete surface or a manifold which is non-linear. Therefore, the Euclidean measure is inappropriate when the input data space has a curvature. Some other similarity measure must be introduced. Our problem is different from the problem of finding a curvature distance between two points on a manifold whose function is known. In our case, the input data are scattered and formed clouds of data. The scattering function is not known and not defined in advance. This makes the problem of measuring the curvature distance between any two vectors more difficult than measuring the distance on a manifold with a

known function such as those in the Riemanian space. A manifold distance which is the shortest distance along the manifold is applied to identify the similarity among the data vectors and then separate them into the same cluster. Since it is difficult to evaluate the manifold distance along unknown surface, we propose a new algorithm to estimate a manifold distance to measure the similarity among input vectors. The approximation criterion can be calculated without knowing the surface function and can be generalized to a high dimensional curved space.

The exact manifold distance can be approximated by a sequence of short hops between neighboring data vectors. Our manifold distance is developed from the algorithm proposed in [8]. The algorithm for approximating manifold distances can be summarized as follows. Point $\mathbf{x}_j$ is the neighbor of point $\mathbf{x}_i$ if the Euclidean distance $||\mathbf{x}_i - \mathbf{x}_j|| \leq \epsilon$, where ϵ is a constant or $\mathbf{x}_j$ is one of the K nearest neighbor points.

Algorithm for Estimating Manifold Distance

1. **For** every $\mathbf{x}_i$ **Do**
2. determine all neighbors of $\mathbf{x}_i$.
3. **EndFor**
4. Construct a weighted undirected graph such that vertex i corresponds to $\mathbf{x}_i$ and an edge (i, j) if and only if $\mathbf{x}_i$ and $\mathbf{x}_j$ are neighbors and its weight is $d_{i,j}$.
5. Find the sequence of points denoted by $\{\mathbf{P_k}\}_{k=0}^{n}$ with $\mathbf{P_0} = \mathbf{x}_i$ and $\mathbf{P_n} = \mathbf{x}_j$ such that the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ is minimal.
6. Find the discrete curve specified by sequence $\{\mathbf{Q_j}\}_{j=0}^{n+1}$ of points where $\mathbf{Q}_0 = \mathbf{P}_0$, $\mathbf{Q}_1 =$ (midpoint of $\mathbf{P}_0$ and $\mathbf{P}_1$), $\mathbf{Q}_2 =$ (midpoint of $\mathbf{P}_1$ and $\mathbf{P}_2$) , ... ,

$\mathbf{Q}_n$ = (midpoint of $\mathbf{P}_{n-1}$ and $\mathbf{P}_n$), $\mathbf{Q}_{n+1} = \mathbf{P}_n$.

7. Calculate an estimation of manifold distance by $d_{\text{manifold}} = \sum_{k=0}^{n} d_{\text{U}}(\mathbf{Q}_k, \mathbf{Q}_{k+1})$ where $d_{\text{U}}(\mathbf{Q}_k, \mathbf{Q}_{k+1})$ is the Euclidean distance between $\mathbf{Q}_k$ and $\mathbf{Q}_{k+1}$.

The shortest distance between vertices i and j in the graph is the estimated manifold distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. The approximation of the manifold distance between two points on the data space is demonstrated in Figure 4.1. Obviously, this distance is different from the Euclidean distance (the straight line) as shown in Figure 4.1.

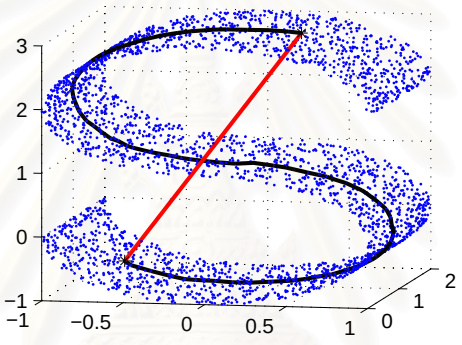


Figure 4.1: Euclidean distance(straight line) and manifold distance(curved line).

Let $d_M(\mathbf{x}, \mathbf{y})$ be the manifold distance and $d_G(\mathbf{x}, \mathbf{y})$ be the graph distance between points $\mathbf{x}$ and $\mathbf{y}$. The proof in [9] shows that for a sufficiently high density (α) of data points, we can always choose a neighborhood size (ϵ and K) large enough so that the graph will have a path not much longer than the true manifold distance, but small enough to prevent the edges that "short circuit" the true geometry of the manifold. More precisely, given arbitrarily small values of $\lambda_1, \lambda_2, \mu > 0$, we can guarantee that with probability at least $1 - \mu$, the estimates

of the following form

$$(1 - \lambda_1)d_M(\mathbf{x}, \mathbf{y}) \leq d_G(\mathbf{x}, \mathbf{y}) \leq (1 + \lambda_2)d_M(\mathbf{x}, \mathbf{y}) \tag{4.1}$$

will hold uniformly over all pairs of data points $\mathbf{x}$,$\mathbf{y}$.

4.2 Weight Adjusting in Non-Euclidean Space and Spherical Mapping

The purpose of weight adjusting in a self-organizing network is to place all prototype neurons at their appropriate locations in order to represent their corresponding clusters. A cluster is represented by a prototype neuron. Usually, the weight adjusting procedure is based on the minimum Euclidean distance between a prototype neuron and an input training vector. The adjusted prototype neuron is moved toward the input training vector and, eventually, located at the centroid of the corresponding cluster. If the input space is a curved space, this algorithm is not appropriate because the prototype neurons cannot be moved along the actual surface of the space. The process of weight adjusting in a non-Euclidean space is similar to that of a Euclidean space. The only differences are the criteria to select a winner and the adjusted weights moved on the surface or manifold. A manifold distance is used to select a winner. A mesh covering the data space is formed by connecting all neighboring neurons. In classical SOM, a neighbor of any neuron is determined from its coordinates on a 2-dimensional map. This mapping scheme cannot be applied to the non-Euclidean case since there are two possible curved distances for any neuron and its neighbor. In this paper, a spherical mapping scheme where all neurons are mapped to a sphere called a *neighborhood sphere*

is introduced to resolve this feasibility. This concept is different from the concept in [10]. In Spherical SOM [10], neurons must be arranged in the spherical lattice before training for considering the neighbors and the position of neurons are fixed through training. In our model, the geometrical structure of data space will be considered so the map should not be fixed position and it can be changed according to the structure of data space. The concept of weight adjusting in a non-Euclidean data space is as follows.

Let $\{\mathbf{x}_i \mid 1 \leq i \leq N\}$ be a finite set of N input data, whose elements will be referred to as a "data vector". The distance between two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ on manifold M is defined by the manifold metric d_M as follows:

$$d_M(\mathbf{x}_i, \mathbf{x}_j) = \inf_{\gamma} \{\text{length}(\gamma)\} \tag{4.2}$$

where γ is in the set of smooth arcs connecting $\mathbf{x}_i$ to $\mathbf{x}_j$ in the manifold. This distance is used to find the winning neuron. The location of neuron j is captured by its synaptic prototype vector $\mathbf{w}_j$. Suppose there are k neurons. Each neuron j is represented by a node Ω_j on the sphere. Let $\{\Omega_j \mid 1 \leq j \leq k\}$ be a set of neuron nodes. Each of Ω_j contains the prototype vectors $\mathbf{w}_j$, whose dimension is equal to the data vectors $\mathbf{x}_i$. Let $\mathbf{w}_{i*}$ be the weight of the winning neuron. The value of $\mathbf{w}_{i*}$ with respect to data vector $\mathbf{x}_a$ is determined by the condition:

$$d_M(\mathbf{w}_{i*}, \mathbf{x}_a) = \min_j d_M(\mathbf{x}_a, \mathbf{w}_j), \quad 1 \leq j, i* \leq k \tag{4.3}$$

The algorithm of weight adjusting for a self-organizing artificial neural network in a non-Euclidean space can be demonstrated as follows:

Proposed Weight Updating Algorithm

1. Initialize the weight vector $\mathbf{w}_1$ selected from data vectors. Create node Ω_1 and set $k = 1$.

2. For each vector $\mathbf{x}_a$, find the winning neuron i^* by using the manifold distance criterion in equation (4.3)

   (a) If $d_M(\mathbf{x}_a, \mathbf{w}_{i*}) > \epsilon$, then create the new node, $k \Leftarrow k + 1, \mathbf{w}_k = \mathbf{x}_a$, $\Omega_k \Leftarrow \Omega_k \cup \{\mathbf{x}_a\}$;

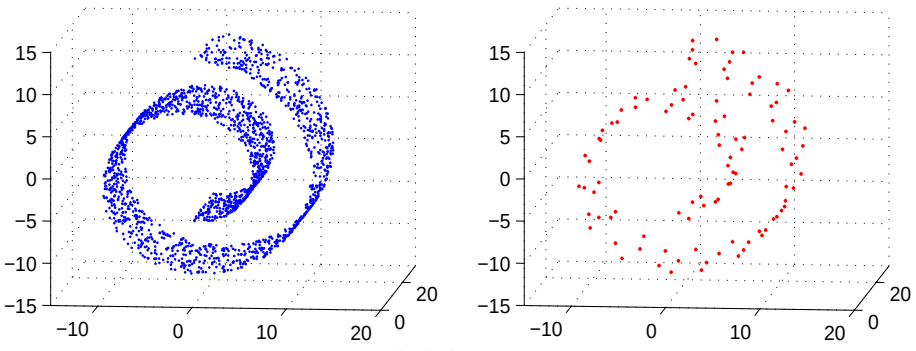
   (b) Otherwise $\Omega_{i*} \Leftarrow \Omega_{i*} \cup \{\mathbf{x}_a\}$.

3. For each Ω_k, compute $a = \arg\min_{\mathbf{x}_i \in \Omega_k} \left(\sum_{\mathbf{x}_j \in \Omega_k} d_M(\mathbf{x}_i, \mathbf{x}_j) \right)$ and then update weight by $\mathbf{w}_k^{new} = \mathbf{x}_a$.

4. Repeat steps 2 to 4 until the solutions can be regarded as steady.

Figure 4.2 shows an example of a 3-dimensional spiral strip consisting of a set of scattered data and the prototype neurons. Notice that all the prototype neurons are located within the data vectors.

Next, a mesh is formed to cover these data vectors by connecting the neighboring neurons. Since each neuron is mapped onto a sphere, the locations of two nearest neurons on the sphere must be determined. First, each prototype neuron in the data space must be mapped to a location on the surface of the *neighborhood sphere*. Then, based on these locations, the neighboring neurons are searched. Let D_{ij} be a minimum manifold distance of two neurons i and j measured in the data space. This D_{ij} is used as an estimated distance on the *neighborhood sphere*. We assume that D_{ij} is the spherical distance lying along the great circle of the sphere



(a) (b)

Figure 4.2: An example of applying weight updating algorithm to a 3-dimensional spiral strip. (a) Data vectors of the spiral strip. (b) The prototype neurons produced by the weight updating algorithm.

as shown in Figure (4.3). Let r be the radius of the circle and $\mathbf{s}_i$ be the location of neuron i on the sphere. Note that $\mathbf{s}_i$ is a vector in a 3-dimensional space.

To find the positions of neurons on a two-dimensional sphere, we apply Multidimensional Scaling in [8] by changing the Multidimensional Scaling procedure [8] for finding the scalar product matrix $\tau(D)$ in the following steps.

$$\mathbf{s}_i \cdot \mathbf{s}_j = ||\mathbf{s}_i||||\mathbf{s}_j|| \cos(\theta) \tag{4.4}$$

where $\mathbf{s}_i$, $\mathbf{s}_j$ are positions on a great circle of sphere of radius r and θ is an angle between $\mathbf{s}_i$ and $\mathbf{s}_j$. Since $||\mathbf{s}_i|| = ||\mathbf{s}_j|| = r$, we have

$$\mathbf{s}_i \cdot \mathbf{s}_j = r^2 \cos(\theta) \tag{4.5}$$

If D_{ij} is the arc length between $\mathbf{s}_i$ and $\mathbf{s}_j$ on great circle of radius r as shown in

Figure 4.3 then we have $\theta = \frac{D_{ij}}{r}$ and replace θ in equation 4.5 to obtain:

$$\mathbf{s}_i \cdot \mathbf{s}_j = r^2 \cos\left(\frac{D_{ij}}{r}\right) \tag{4.6}$$

The value of $\mathbf{s}_i \cdot \mathbf{s}_j$ will be used to map a prototype neuron onto its neighborhood sphere by this algorithm.

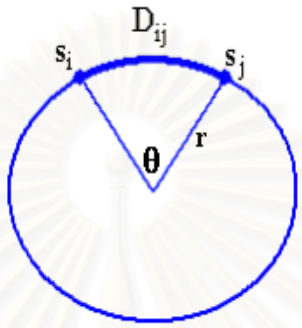


Figure 4.3: A great circle of radius r.

Algorithm for Mapping Prototype Neurons to Neighborhood Sphere

1. Approximate the radius of sphere $r = \max_{i,j}\left(\frac{D_{ij}}{\pi}\right)$.
2. Set matrix $\tau(D) = r^2 \cos\left(\frac{D_{ij}}{r}\right)$.
3. Compute the location matrix of all neurons by solving $[\mathbf{s}_i^T \mathbf{s}_j] = \tau(D)$ using these steps:
   (a) let λ_p be the p^{th} eigenvalue of the matrix $\tau(D)$
   (b) let ν_p be the p^{th} eigenvector of λ_p
   (c) set the p^{th} row of the location matrix to $\sqrt{\lambda_p}\nu_p$

4. Keep the first three rows of the location matrix and assign each normalized column of the location matrix j to each $\mathbf{s}_j$.

Example Given distance matrix D as followed:

$$D = \begin{bmatrix} 0 & 1.0000 & 1.0000 & 1.4142 \\ 1.0000 & 0 & 1.4142 & 1.0000 \\ 1.0000 & 1.4142 & 0 & 1.0000 \\ 1.4142 & 1.0000 & 1.0000 & 0 \end{bmatrix}$$

1. Approximate radius of sphere $r = \max_{i,j} \left(\frac{D_{ij}}{\pi} \right)$.

2.
$$\tau(D) = r^2 \cos\left(\frac{D_{ij}}{r} \right) = \begin{bmatrix} 0.2026 & -0.1227 & -0.1227 & -0.2026 \\ -0.1227 & 0.2026 & -0.2026 & -0.1227 \\ -0.1227 & -0.2026 & 0.2026 & -0.1227 \\ -0.2026 & -0.1227 & -0.1227 & 0.2026 \end{bmatrix}$$

3. Compute the location matrix such that $[\mathbf{s}_i^T \mathbf{s}_j] = \tau(D)$:

   (a) Find eigenvalues of the matrix $\tau(D)$.

   $$\lambda_1 = 0.4053,\ \lambda_2 = 0.4053,\ \lambda_3 = 0.2455,\ \lambda_4 = -0.2455$$

   (b) Find eigenvectors of the matrix $\tau(D)$.

   $$\nu_1 = \begin{bmatrix} 0.7071 \\ -0.0000 \\ 0.0000 \\ -0.7071 \end{bmatrix}, \nu_2 = \begin{bmatrix} 0.0000 \\ 0.7071 \\ -0.7071 \\ 0 \end{bmatrix}, \nu_3 = \begin{bmatrix} 0.5000 \\ -0.5000 \\ -0.5000 \\ 0.5000 \end{bmatrix}, \nu_4 = \begin{bmatrix} -0.5000 \\ -0.5000 \\ -0.5000 \\ -0.5000 \end{bmatrix}$$

(c)

$$\begin{bmatrix} \sqrt{\lambda_1}\nu_1 \\ \sqrt{\lambda_2}\nu_2 \\ \sqrt{\lambda_3}\nu_3 \\ \sqrt{\lambda_4}\nu_4 \end{bmatrix} = \begin{bmatrix} 0.4502 & -0.0000 & 0.0000 & -0.4502 \\ 0.0000 & 0.4502 & -0.4502 & 0 \\ 0.2477 & -0.2477 & -0.2477 & 0.2477 \\ -0.2477i & -0.2477i & -0.2477i & -0.2477i \end{bmatrix}$$

4. Choose the first three rows of the matrix in 3(c),

$$\begin{bmatrix} 0.4502 & -0.0000 & 0.0000 & -0.4502 \\ 0.0000 & 0.4502 & -0.4502 & 0 \\ 0.2477 & -0.2477 & -0.2477 & 0.2477 \end{bmatrix}$$

$$s_1 = \begin{bmatrix} 0.4502 \\ 0.0000 \\ 0.2477 \end{bmatrix}, \; s_2 = \begin{bmatrix} -0.0000 \\ 0.4502 \\ -0.2477 \end{bmatrix}, \; s_3 = \begin{bmatrix} 0.0000 \\ -0.4502 \\ -0.2477 \end{bmatrix}, \; s_4 = \begin{bmatrix} -0.4502 \\ 0 \\ 0.2477 \end{bmatrix}$$

and normalize each vector to have unit length:

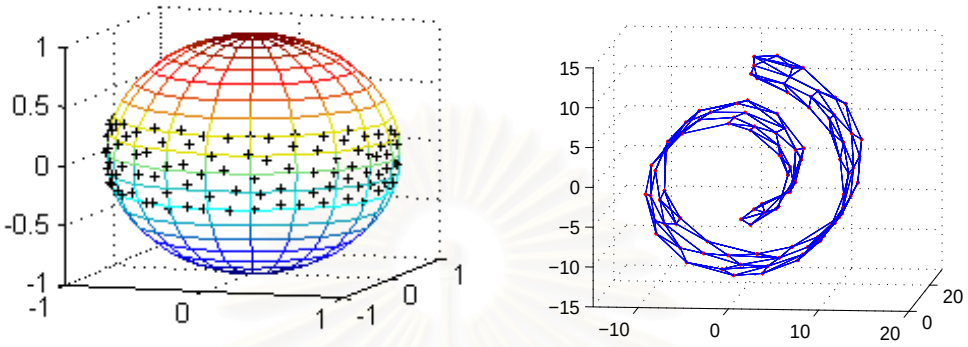
$$s_1 = \frac{s_1}{\|s_1\|} = \begin{bmatrix} 0.8761 \\ 0.0000 \\ 0.4821 \end{bmatrix}, \; s_2 = \frac{s_2}{\|s_2\|} = \begin{bmatrix} -0.0000 \\ 0.8761 \\ -0.4821 \end{bmatrix}$$

$$s_3 = \frac{s_3}{\|s_3\|} = \begin{bmatrix} 0.0000 \\ -0.8761 \\ -0.4821 \end{bmatrix}, \; s_4 = \frac{s_4}{\|s_4\|} = \begin{bmatrix} -0.8761 \\ 0 \\ 0.4821 \end{bmatrix}$$

where $\|\cdot\|$ is the Euclidean norm.

Thus s_1, s_2, s_3, and s_4 are the positions of neurons in the output space.

The locations of all prototype neurons in Figure 4.2 on the *neighborhood sphere* are denoted in Figure 4.4(a). Figure 4.4(b) shows the mesh formed by the connections of all prototype neurons.



(a) (b)

Figure 4.4: The *neighborhood sphere* of the data space in Figure 4.2. (a) The locations of all prototype neurons on the sphere. (b) The feature map of the prototype neurons.

CHAPTER 5

Experimental Results

5.1 Discrete Shortest Curve on Manifold

A manifold distance is the length of the shortest curve between two points on the manifold. The exact manifold distance can be estimated by a sequence of short hops between neighboring data vectors. In this experiment, we use 24 nearest neighbor points to construct a graph and apply our algorithm for approximating the shortest curve on a discrete surface of a cylinder. The results of our model are shown in Figures 5.1 (a), (c), and (e) compared with the model in [8] shown in Figures 5.1 (b), (d), and (f). For the other experiment, we use eight nearest neighbor points to construct a graph and apply our model to estimate the shortest curve on noisy Swissroll data and spherical data. The results indicate that the approximation of the shortest curve on manifold of our algorithm is smoother and more robust with noisy data than the algorithm in [8] as shown in Figures 5.2 (a) and (c) compared with the algorithm in [8] as shown in Figures 5.2 (b) and (d).

5.2 Prototype Selection

Prototype selection concerns the method of finding representative elements from a given data set. It is necessary for several applications such as pattern classification, clustering, data compression, data analysis, data mining, image segmentation, and others. Most of all existing prototype selection schemes are not suitable for large data set on or near a curved space such as a surface or a manifold. In this experiment, we show that our method can be used for finding representative elements from data set in a curved space. We use the synthetic data randomly generated by *sinusoidal* function and *spiral* function to be the test data as shown in Figures 5.3(a) and 5.3(d), respectively.

We apply our algorithm and SOM algorithm to find the representative elements from the synthetic data and set ϵ to 4 for the data generated by *sinusoidal* function and ϵ to 15 for the data generated by *spiral* function. After training, the result from our algorithm as presented in Figures 5.3(c) and 5.3(f). We initialize four weight vectors for the data generated by *sinusoidal* function and initialize five weight vectors for the data generated by *spiral* function to SOM algorithm and the results from SOM algorithm are shown in Figures 5.3(b) and 5.3(e). In the *sinusoidal* function, our algorithm and SOM give the same results. But in the *spiral* function, our algorithm correctly clusters the data according to their natural distribution while SOM does not preserve the natural distribution of the data in each cluster.

5.3 Data Compression

Data Compression is an important problem in the data preprocessing to reduce a large data set. In this section, we will compare the result of our algorithm with the conventional SOM algorithm and show that the performance of our algorithm is better than the conventional SOM algorithm. Especially, when the data space is a curved space such as a surface or a manifold. In the experiment, we use Swissroll data shown in Figure 5.4 which is the 2-dimensional manifold embedded in a 3-dimensional space to test our model and compare the result with SOM. We use 10 nearest neighbor points to construct the graph for estimating the manifold distance and set ϵ to 3.73 in weight adjusting step. After six epochs, we get 100 weight vectors for representing the input data as shown in Figure 5.5(b). After that we initiate 100 weight vectors by random from the input data for SOM algorithm and train the same data for 1000 epochs. After training, the prototype neurons representing the input data as presented in Figure 5.5(a) are obtained. Furthermore, we demonstrate that SOM does not preserve the feature map in a curved space. Another experimental example of a spherical data set in a closed discrete surface is given in Figure 5.6. In this experiment, we set the parameter ϵ to 0.32 in weight adjusting step and construct the graph with 8 nearest neighbor points. After training by our algorithm, there are 100 weight vectors representing the data set as demonstrated in Figure 5.7(b). In classical SOM algorithm, 100 neurons are trained and the result is presented in Figure 5.7(a). Clearly, when the input data are on a closed surface, the marginal neurons on one side of SOM map cannot be the neighbors of the neurons on the opposite side. This creates an

incomplete and disconnected mesh. But the mesh generated by our algorithm is completely connected.

5.4 Data Clustering

Clustering techniques are important tools for a variety of scientific area such as data mining, pattern recognition, image segmentation, statistical data analysis and others. Its goal is to partition N data vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ into the k clusters with respect to a distance or a similarity measure. Although the clustering techniques have been widely applied in several scientific fields, it always deploys conventional Euclidean distance to measure the similarity between the data vectors. This conventional distance cannot be applied to measure the similarity between the data vectors in a curved space such as a manifold. In this experiment, we partition the Swissroll data into two, three, four, and five clusters by setting ϵ to 40, 30, 25, and 22.5, respectively. The experimental results are shown in Figures 5.8 (a), (b), (c), and (d), respectively. Each cluster is painted with dotted patterns. Figure 5.9 shows the classification results using classical SOM. Note that when the number of clusters increases the data are not classified along their natural surface. The second clustering test set is shown in Figure 5.10(a). This test set is trained by our model with ϵ equals 10 and four nearest neighbors in the step of the manifold distance approximation. The experimental results are shown in Figures 5.10(b) and 5.10(c) and compared with SOM in Figure 5.10(d). Although both classical SOM and ours can classify the data into two clusters, the locations of the prototype neurons of SOM are not located inside the clusters. The third test set of clustering is shown in Figure 5.11(a). This test

set is trained by our model with ϵ equals 8 and 4-nearest neighbors in the step of the manifold distance approximation. The experimental results are shown in Figure 5.11(b) and compared with SOM in Figure 5.11(c). The fourth test set of clustering is shown in Figure 5.12(a). This test set is trained by our model with ϵ to 5 and 10-nearest neighbors in the step of the manifold distance approximation. The experimental results are shown in Figure 5.12(b) and compared with SOM in Figure 5.12(c). Another experiment, we partition two spiral data set into two class compared the result with SOM as showed in Figure 5.13 (a) and (b).

5.5 Piecewise Linear Skeletonization on Manifold

Skeletonization is one of the fundamental problems in image processing and computer vision. In this section, we apply our approach to find the skeleton of a surface. First, we use our approach to partition the data into the different classes in order to get the different prototype vectors on a surface. Second, Prim's Algorithm is used to find the minimum spanning tree for these prototype vectors and connect these vectors with a discrete curve using the algorithm for estimating the manifold distance. This discrete curve that passes through these prototype vectors is also the skeleton of the surface. We apply this concept to find the skeleton of Swissroll data and S-curved data with 50% noise and 75% noise. To set the parameter, the number neighboring points for the original data and the data with 50% noise is set to eight while that of the data with 75% noise is set to four. The results are shown in Figures 5.14 and 5.15.

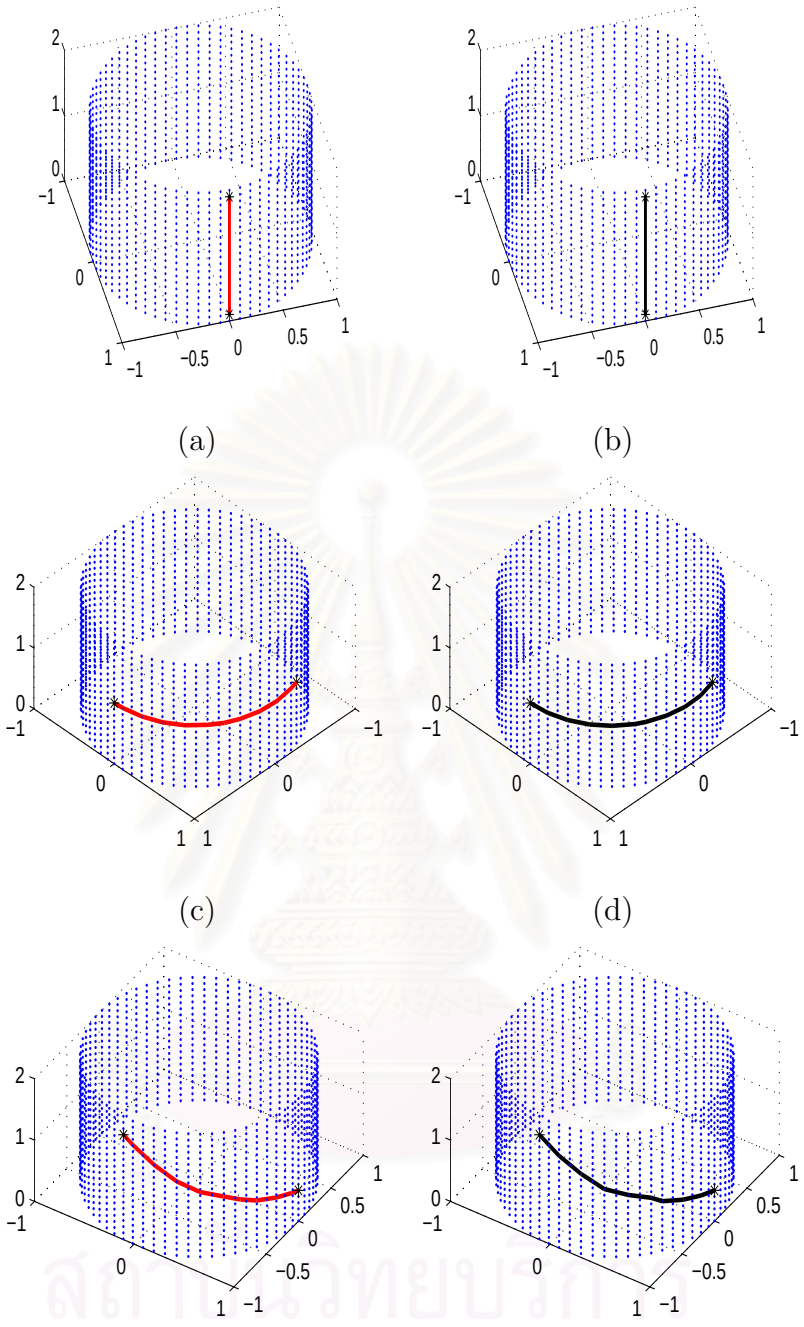


Figure 5.1: (a), (c) and (e) show the manifold distance by our algorithm. (b), (d) and (f) show the manifold distance by the algorithm in [8].

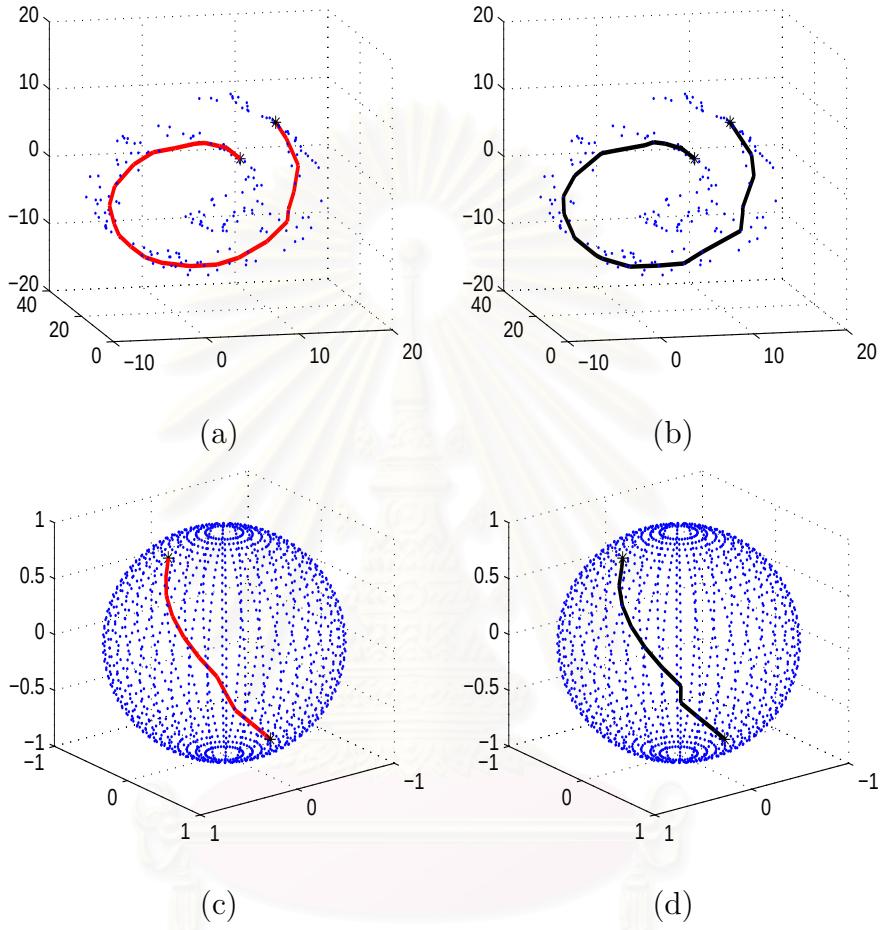


Figure 5.2: (a) and (c) Manifold distance by our algorithm. (b) and (d) Manifold distance by the algorithm in [8].

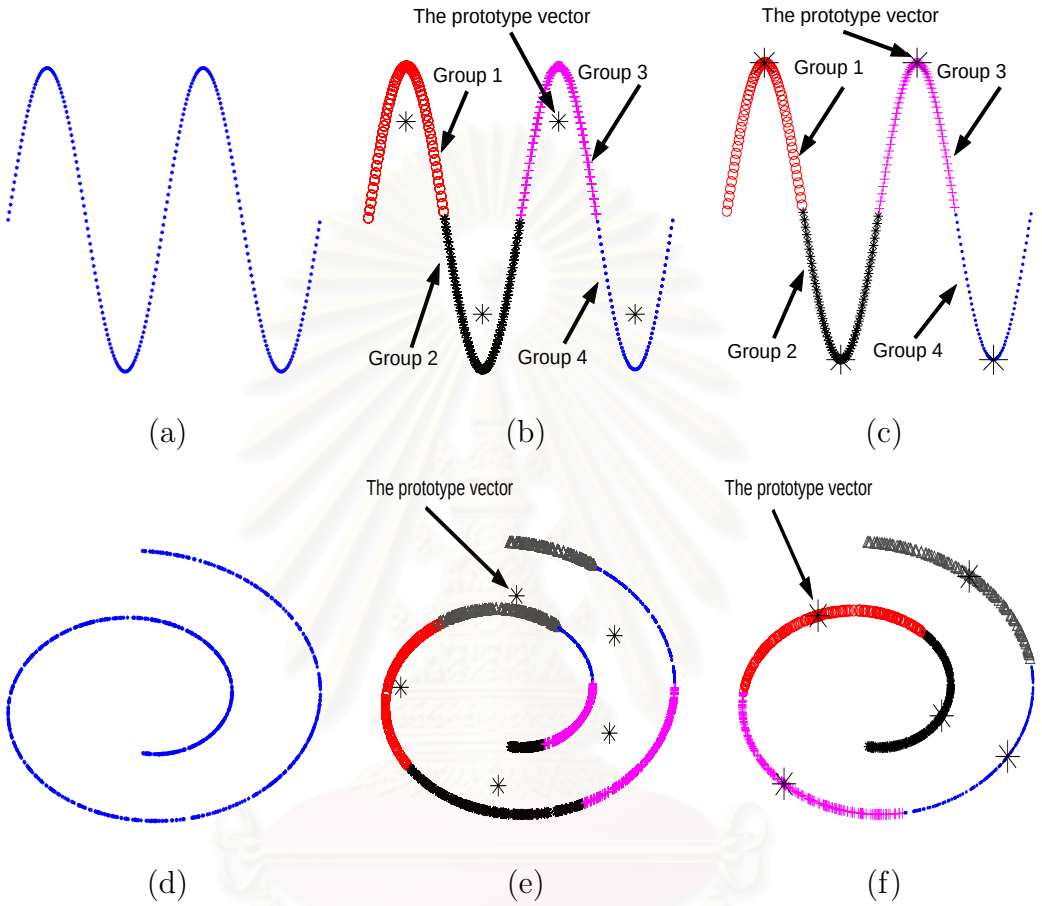


Figure 5.3: (a) and (d) show manifold data. (b) and (e) show prototype vectors (star points) trained by SOM. (c) and (f) show prototype vectors (star points) trained by our model.

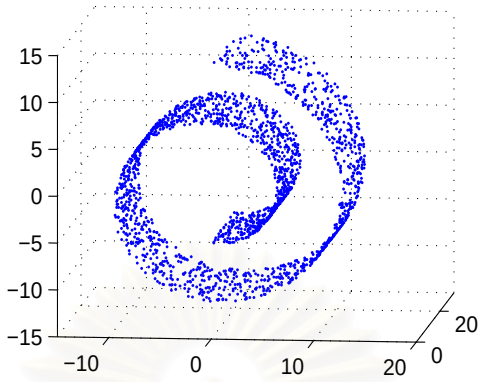
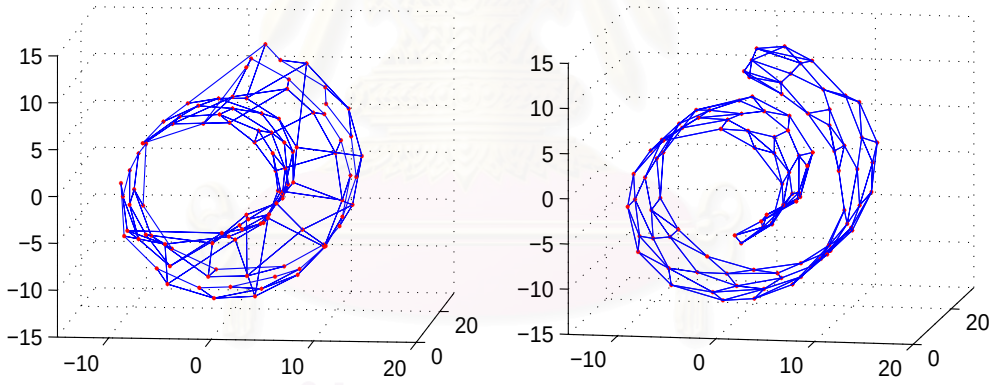


Figure 5.4: Swissroll data plotted in a 3-dimensional space.



(a) (b)

Figure 5.5: Two different meshes and mappings of the swissroll. (a) Mesh trained by SOM. (b) Mesh trained by our algorithm.

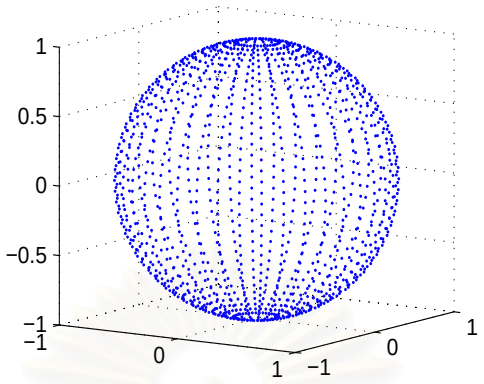
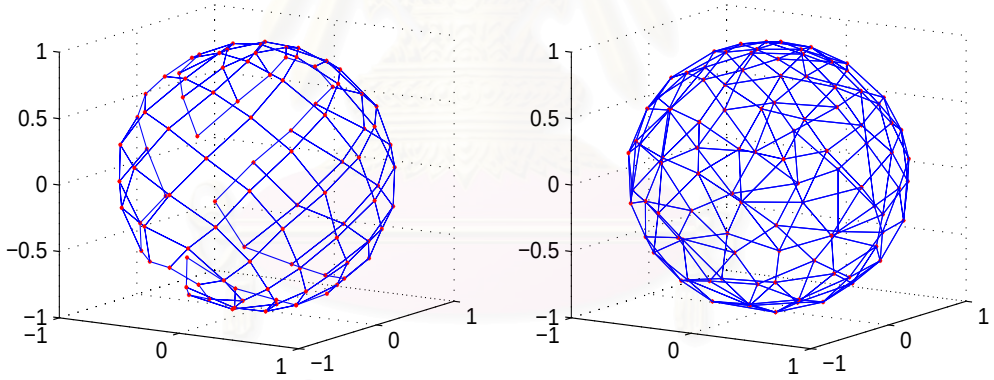


Figure 5.6: An example of input data scattered in a spherical space.



(a) (b)

Figure 5.7: An example of data scattered on a spherical space. (a) Mesh generated by SOM. (b) Mesh generated by our algorithm.

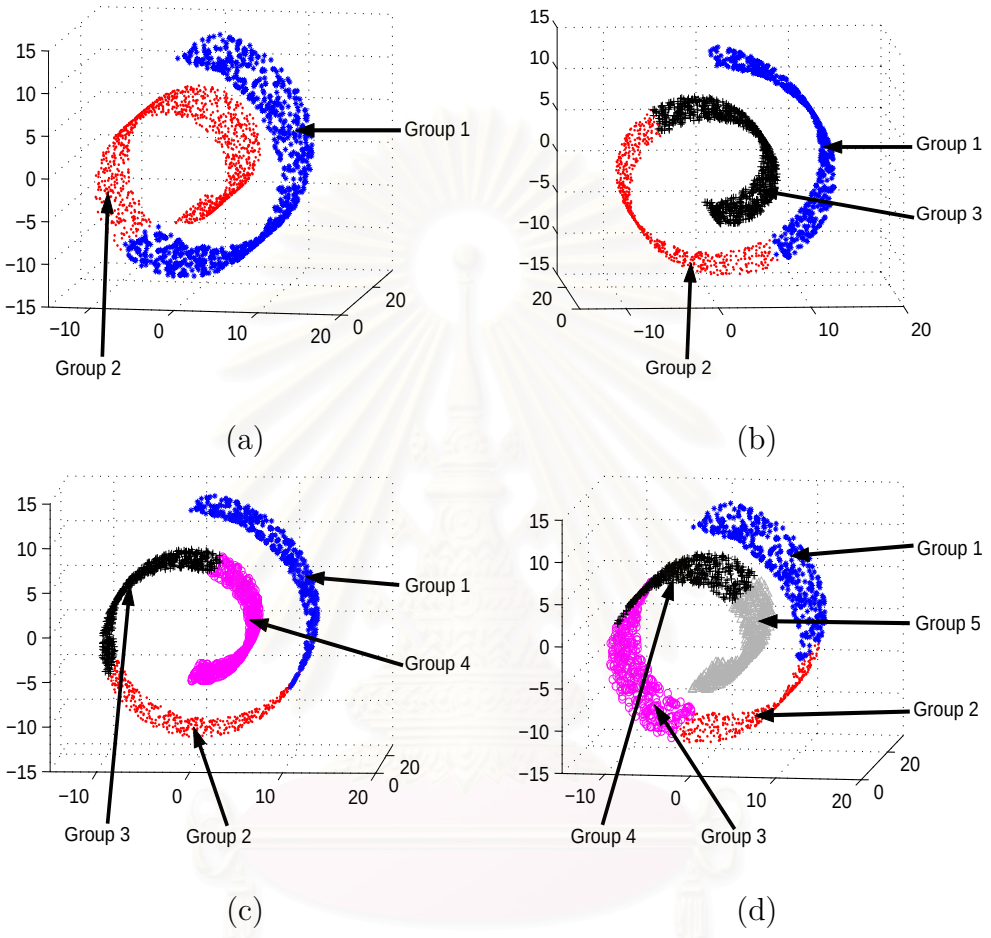


Figure 5.8: An example data clustering based on our Algorithm. (a) Two clusters. (b) Three clusters. (c) Four cluster. (d) Five clusters.

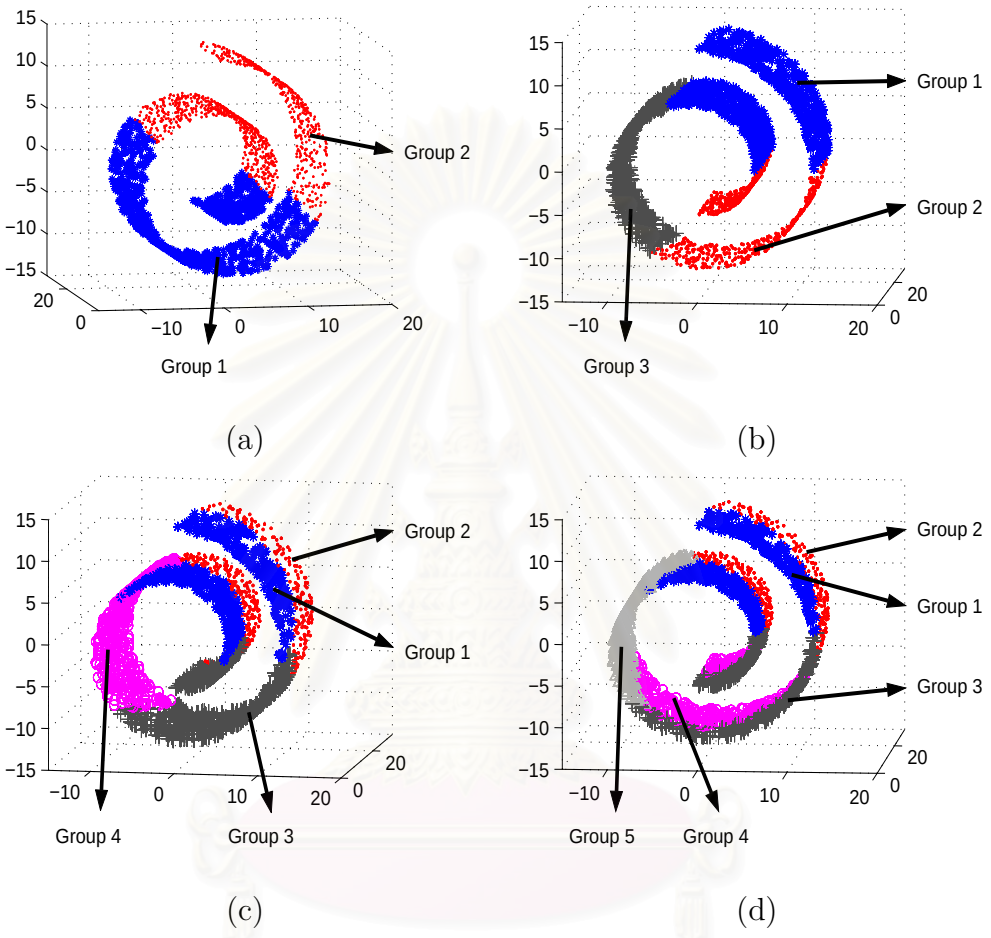


Figure 5.9: An example data clustering based on SOM Algorithm. (a) Two clusters. (b) Three clusters. (c) Four cluster. (d) Five clusters.

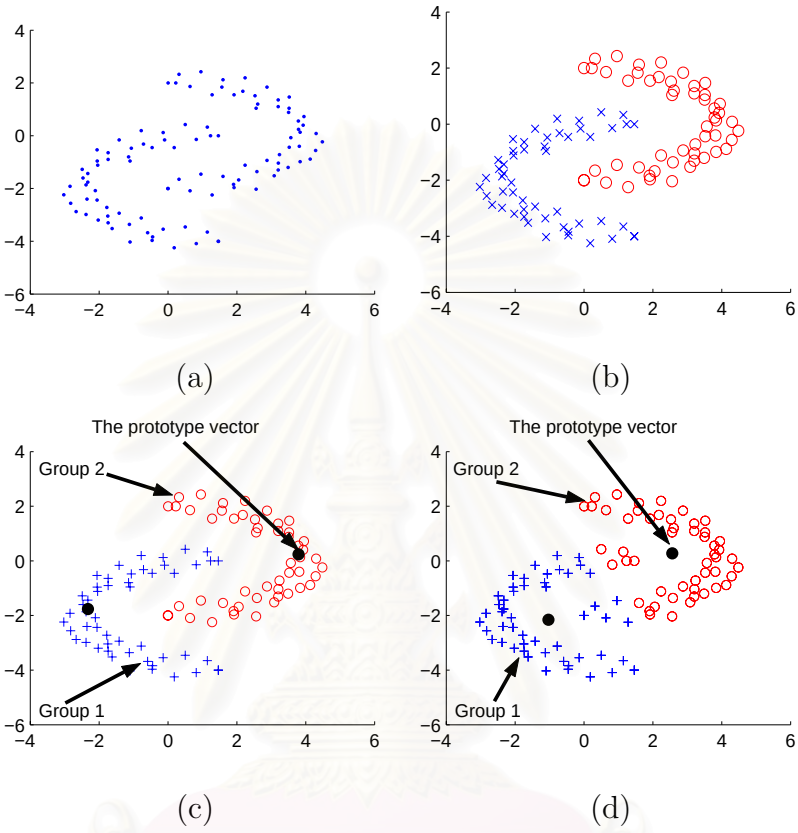
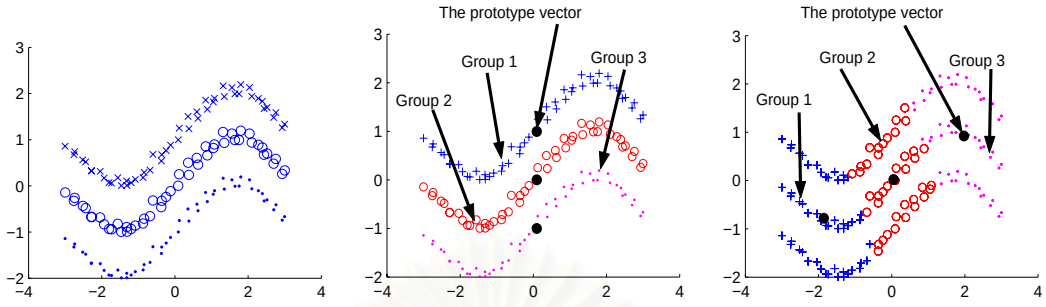
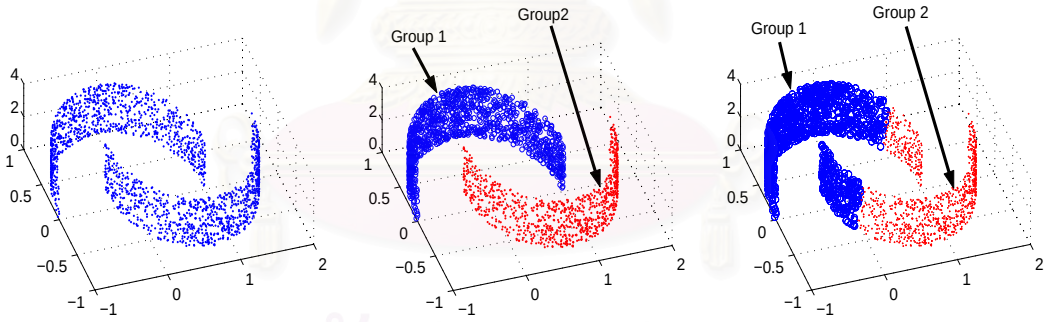


Figure 5.10: An example of clustering a 2-dimensional data set. (a) The input data. (b) Two possible clusters. (c) Two clusters generated by our model with two prototype neurons (black points). (d) Two clusters generated SOM with two prototype neurons (black points).



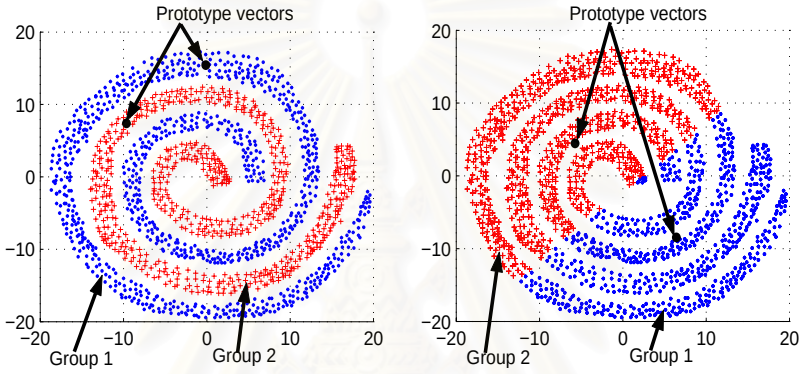
(a) (b) (c)

Figure 5.11: (a) The input data. (b) Three clusters of our model with three weights (black point). (c) Three clusters of SOM with three weights (black point).



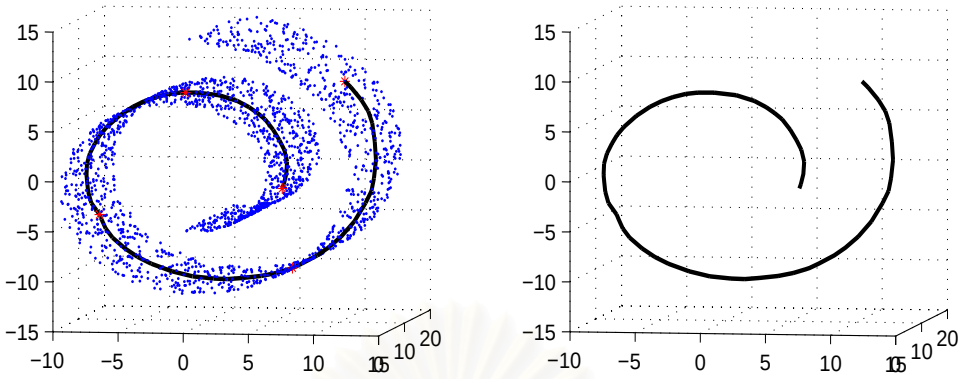
(a) (b) (c)

Figure 5.12: (a) The input data. (b) Two clusters of our model. (c) Two clusters of SOM.

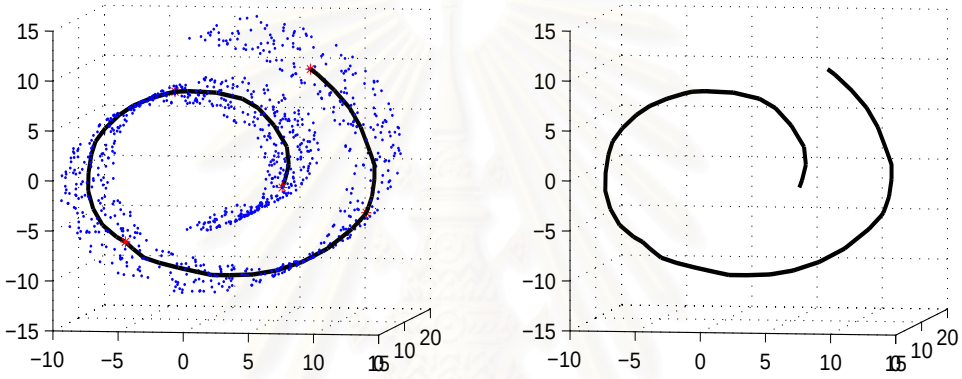


(a) (b)

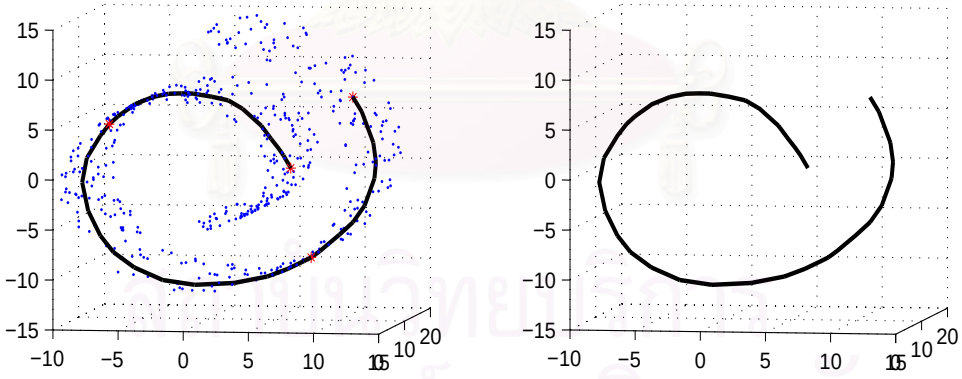
Figure 5.13: (a) two clusters trained by our Algorithm. (b) two clusters trained by SOM. Each group is illustrated by different color interesting.



(a) The original data.

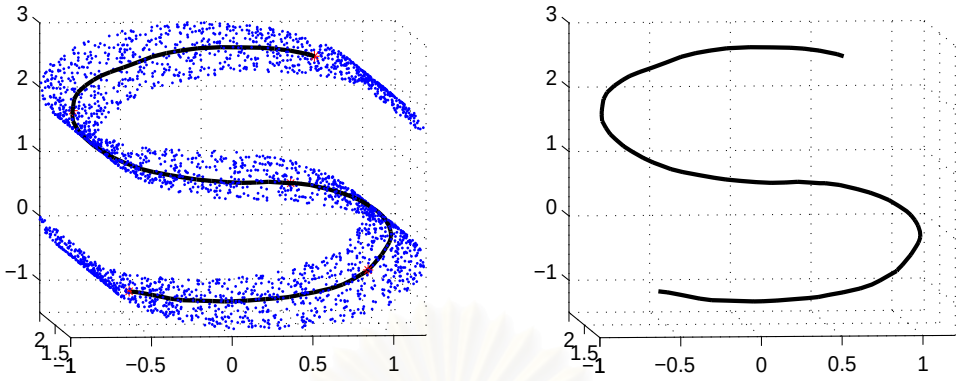


(b) 50% noise.

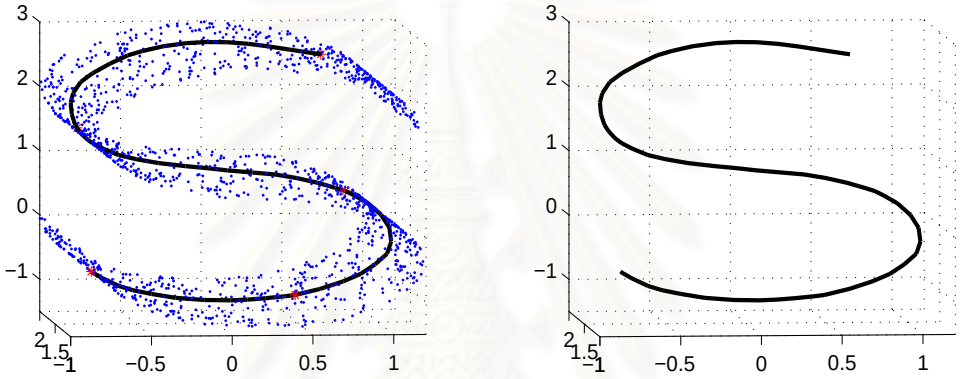


(c) 75% noise.

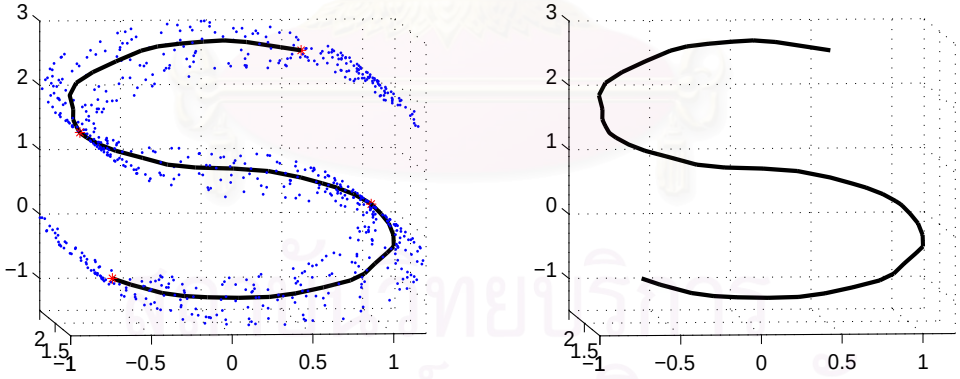
Figure 5.14: The skeleton of the surface.



(a) The original data.



(b) 50% noise.



(c) 75% noise.

Figure 5.15: The skeleton of the surface.

CHAPTER 6

CONCLUSION

This research presents a novel unsupervised competitive learning algorithm to adjust a weight of neurons for a Self-Organizing Artificial Neural Network in a non-Euclidean space. A novel method bases on the competitive learning using an estimation of manifold distance, the shortest distance on a manifold, to measure the similarity between weight vectors and data vectors and partition the data into the k cluster base on a distance on a manifold. This distance can be computed without a function of a surface or manifold. Furthermore, we present a new mapping of neurons which is more appropriate to a curved space than the traditional mapping in SOM model. A new criterion suits for several problems such as feature extraction, pattern classification, and data compression when the actual surface distance is most the critical factor. However, the drawback of our algorithm still exists in the step of an approximation of manifold distance which may be lead to errors in the algorithm. If the neighborhood is too large with respect to folds in the manifold, it can lead to short-circuit errors. But, if the neighborhood is very small, it can fragment the input data into a large number of disconnected regions. So, the achievement of this algorithm depends on the ability of choosing a neighbor size (ϵ or K) in the algorithm for estimating of manifold distance.

REFERENCES

[1] T. Kohonen. "Self-Organizing Maps", *Springer Series in Information Sciences, 3rd edition*, 2001.

[2] Pasi Koikkalainen and Erkki, "Self-organizing hierarchical feature maps". *In Proc, Of the IJCNN 1990*, volume II, pages 279-285, 1990.

[3] Teuvo Kohonen, Samuel Kaski, and Harri Lappalainen, "Self-organized formation of various invariant-feature filters in the adaptive-subspace som", *Neural Computation*, 9(6):1321-1344, 1997.

[4] Christopher M. Bishop, Markus Svenson, and Christopher K.I. Williams, "GTM: The generative topographic mapping", *Neural Computation*, 10(1): 215-234, 1998.

[5] Thore Graepel and Klaus Obermayer, "A stochastic self-organizing map for proximity data", *Neural Computation*, 11(1): 139-155, 1999.

[6] Da Deng, Nikola Kasabov, "ESOM: An Algorithm to Evolve Self-Organizing Maps from On-Line Data Streams", *IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)*, Volume 6, pp. 6003, July 24 - 27, 2000.

[7] P. Demartines and J. Herault, "Curvilinear Component Analysis: A self-organizing neural network for nonlinear mapping of data sets", *IEEE Transaction on Neural Networks*, 8(1):148-154, January 1997.

[8] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction", *Science*, 290(5500):2319-2323, December 2000.

[9] Mira Bernstein, Vin de Silva, John C. Langford, and Joshua B. Tenenbaum, "Graph approximations to geodesics on embeded manifolds", *Technical Report*, Department of Psychology, Stanford University, 2000.

[10] H. Ritter, "Self-organizing Maps in non-euclidean Spaces", *WSOM 99 Conference Proceedings*, 1999.

[11] Simon Haykin, *NEURAL NETWORKS: A Comprehensive Foundation Second Edition*, Prentice Hall International, Inc, 1999.

VITAE

Saichon Jaiyen received a Bachelor degree in Mathematics from the Department of Mathematics, Faculty of Science, Kasetsart University in 1999.

Publication

S. Jaiyen and C. Lursinsap,"Non-Euclidean Self- Organizing Classification Using Natural Manifold Distance", *IEEE Proceedings of International Joint Conference on Neural Networks (IJCNN'04)*, July 26 - 29, 2004.